JEGYZŐKÖNYV

Mesterséges intelligencia és neurális hálózatok
Genetikus algoritmus

Orosz Kristóf
EYZWG9

Sárospatak
2025. november 27.

Tartalomjegyzék

Bevezetés

A genetikus algoritmus az **útvonaloptimalizálást** végzi, vagyis azt számolja ki, hogy a depók között milyen sorrendben kell a járműnek haladnia ahhoz, hogy a teljes út a lehető legrövidebb legyen.

A működés alapja az, hogy **kezdetben több véletlenszerű útvonalat** generálok (ezek alkotják a populációt), majd ezeket **több generáción keresztül fejlesztem**. Minden egyes útvonalhoz kiszámítom a teljes távolságot az Euklideszi képlet segítségével (tavolsag() és utHossz() függvények).

Ezután **a legjobb, azaz a legrövidebb utak kerülnek előtérbe**, míg a rosszabbak kiszelektálódnak. A jó megoldásokból **új „gyermek" útvonalakat hozok létre** a keresztez() függvény segítségével, amely két meglévő útvonalat kombinál. A mutacio() függvény kis valószínűséggel véletlenül megcserél két depót az útvonalban, ezzel biztosítva a változatosságot és azt, hogy az algoritmus ne ragadjon be egy helyi minimumba.

A folyamat **200 generáción keresztül ismétlődik**, minden körben egyre rövidebb és hatékonyabb útvonalakat találva. A végén az algoritmus visszaadja a **legrövidebb útvonalat és annak hosszát**, vagyis a legoptimálisabb megoldást a depók közti bejárásra.

A projekt genetikus algoritmusa

```
<script>
  // Itt számítom ki a depók közötti euklideszi távolságot.
  function tavolsag(a, b) {
    return Math.sqrt((a.x - b.x) ** 2 + (a.y - b.y) ** 2);
  }

  // Ezzel a függvénnyel kiszámítom egy adott útvonal teljes hosszát.
  function utHossz(ut, raktarak) {
    let osszesHossz = 0;
    for (let i = 0; i < ut.length - 1; i++) {
      const a = raktarak.find(d => d.nev === ut[i]);
      const b = raktarak.find(d => d.nev === ut[i + 1]);
      osszesHossz += tavolsag(a, b);
    }
    const start = raktarak.find(d => d.nev === ut[0]);
    const vege = raktarak.find(d => d.nev === ut[ut.length - 1]);
    osszesHossz += tavolsag(vege, start);
    return osszesHossz;
  }

  // Ezzel a függvénnyel hozom létre az első populációt, amely véletlenszerű útvonalakat
tartalmaz.
  // Minden egyed egy depósorrend, vagyis a jármű egy lehetséges útvonala.
  // Így indul el a keresés, több különböző megoldással egyszerre.
  function letrehozPopulacio(raktarak, meret) {
```

```
    const populacio = [];
    for (let i = 0; i < meret; i++) {
      const egyed = [...raktarak.map(d => d.nev)];
      // A depók sorrendjét véletlenszerűen megkeverem (Fisher–Yates algoritmus)
      for (let j = egyed.length - 1; j > 0; j--) {
        const rand = Math.floor(Math.random() * (j + 1));
        [egyed[j], egyed[rand]] = [egyed[rand], egyed[j]];
      }
      populacio.push(egyed);
    }
    return populacio;
  }

  // Ezzel a függvénnyel rendelem a populációt az útvonalhossz szerint.
  function rendezPopulacio(populacio, raktarak) {
    return populacio.sort((a, b) => utHossz(a, raktarak) - utHossz(b, raktarak));
  }

  // A keresztezés során két szülő útvonalból hozok létre egy új "gyermek" útvonalat.
  // A célom, hogy a jó megoldások tulajdonságai kombinálódjanak, és így egy még jobb útvonal
jöjjön létre.
  function keresztez(sz1, sz2) {
    const start = Math.floor(Math.random() * sz1.length);
    const end = start + Math.floor(Math.random() * (sz1.length - start));
    const resz = sz1.slice(start, end);
    const gyerek = sz2.filter(g => !resz.includes(g));
    return [...resz, ...gyerek];
  }

  // A mutációt arra használom, hogy a genetikus keresés ne ragadjon be egy helyi minimumba.
  function mutacio(egyed, valoszinuseg) {
    const ut = [...egyed];
    if (Math.random() < valoszinuseg) {
      const i = Math.floor(Math.random() * ut.length);
      const j = Math.floor(Math.random() * ut.length);
      [ut[i], ut[j]] = [ut[j], ut[i]];
    }
    return ut;
  }



  // Ez a genetikus algoritmus fő függvénye, amely végrehajtja a teljes evolúciós folyamatot.
  function genetikusAlgoritmus(raktarak, generaciok = 200, populacioMeret = 60) {
    // Létrehozom a kezdeti populációt
    let populacio = letrehozPopulacio(raktarak, populacioMeret);

    // Itt zajlik az evolúciós ciklus – 200 generáción keresztül javítom a megoldásokat
    for (let gen = 0; gen < generaciok; gen++) {
```

```
    // A legjobb útvonalakat előresorolom
    populacio = rendezPopulacio(populacio, raktarak);
    const ujPop = [];

    // A két legjobb útvonalat közvetlenül továbbviszem
    ujPop.push(populacio[0], populacio[1]);
    while (ujPop.length < populacioMeret) {
      const sz1 = populacio[Math.floor(Math.random() * (populacioMeret / 2))];
      const sz2 = populacio[Math.floor(Math.random() * (populacioMeret / 2))];
      let gyerek = keresztez(sz1, sz2);
      gyerek = mutacio(gyerek, 0.2); // 20%-os mutációs eséllyel dolgozom
      ujPop.push(gyerek);
    }

    // A generáció végén az új populáció lesz az aktuális
    populacio = ujPop;
  }

  // Kiválasztom a legjobb (legrövidebb) útvonalat
  const legjobb = rendezPopulacio(populacio, raktarak)[0];
  const tav = utHossz(legjobb, raktarak).toFixed(2);

  // A legjobb útvonalat és annak hosszát visszaadom
  return { legjobb, tav };
}
```